AMENDMENT

Please amend the above-identified application as follows:

Amendments to the Specification:

Please amend the following paragraphs as indicated below:

[0011] **Figure 2** is an exemplary ~~Java~~ JAVA control source (JCS) file in an embodiment of the invention.

[0014] An integrated development environment (IDE), such as WebLogic® Workshop (available from BEA Systems, Inc.), can provide controls (e.g., ~~Java~~ JAVA® controls) that make it easy for users to encapsulate business logic and to access enterprise resources such as databases, legacy applications, and web services. In one embodiment, there can be three different types of Controls: built-in Controls, portal controls, and custom Controls.

[0015] Built-in controls provide easy access to enterprise resources. By way of a non-limiting example, a database control makes it easy to connect to a database and perform operations on the data using simple SQL statements, whereas an EJB control enables users to easily access an EJB. Built-in controls provide simple properties and methods for customizing their behavior, and in many cases users can add methods and callbacks to further customize the control. In one embodiment, a portal control is a kind of built-in ~~Java~~ JAVA control specific to the portal environment. If users are building a portal, users can use portal controls to expose tracking and personalization functions in multi-page portlets.

[0016] In one embodiment, users can also build a custom control from scratch. Custom controls are especially powerful when used to encapsulate business logic in reusable components. It can act as the nerve center of a piece of functionality, implementing the desired overall behavior and delegating subtasks to built-in Controls (and/or other custom controls). This use of a custom ~~Java~~ JAVA control ensures modularity and encapsulation. Web services, JSP pages, or other custom Controls can simply use the custom ~~Java~~ JAVA control to obtain the

desired functionality, and changes that may become necessary can be implemented in one software component instead of many.

[0023] **Figure 1** is [[a]] an illustration of an exemplary graphical representation of a control in an embodiment. After users create a control source (JCS) file in a language such as (but not limited to) ~~Java~~ JAVA®, a graphical canvas **100** can provide a space in which users can create a visual representation of their control's programmatic interface as well as the controls it may itself be using. The left side **102** can display operations that will be visible to the control's clients, while the right side **104** can display nested controls. In one embodiment, users can have easy access to a control's source file by double-clicking the graphical representation of the control.

[0026] In one embodiment, users can define certain IDE characteristics for their ~~Java~~ JAVA control. These include the icon that represents it in palettes and menus (and whether it is displayed in the palette at all), its description in a Property Editor, and so on.

[0027] Once a control has been added to an application, users can invoke its methods using the standard ~~Java~~ JAVA dot notation (assuming the control was written in the ~~Java~~ JAVA programming language). By way of a non-limiting example, assume that the "CustomerDb" ~~Java~~ JAVA control is added to an application and a variable is declared for the control as "custDb", and that the control defines a method as follows:

```
String [] getAllCustomerNames()
```

[0030] In one embodiment, a callback definition in a ~~Java~~ JAVA control may look like the following:

```
void onReportStatus(String status);
```

[0033] The designer of a ~~Java~~ JAVA® control may choose whether or not to explicitly declare that exceptions are thrown by the control's methods. If a control method is declared to

throw exceptions, users can enclose their invocations of that method in a ~~Java~~ JAVA® try-catch block.

[0035] A built-in control can be used by a custom ~~Java~~ JAVA control to delegate subtasks, but it can also be used directly by a web service in much the same way. Built-in controls, as well as custom controls, can furthermore be invoked from a web page, although the procedure for invoking these controls from a web page environment is somewhat different.

[0037] In one embodiment, files with the extension JCX are control extensions for controls written in ~~Java~~ JAVA®. They typically include a collection of method definitions that allow users to easily access a resource such as a database or another enterprise resource. In some cases, users may use an existing JCX file that was produced by another member of their team or another organization. By way of a non-limiting example, if many web services will use the same database, a single author might create a Database control extension (JCX file) that describes the interface to the database. Then multiple web service authors might use that JCX file to create a Database Control in their service and use it to access the common database. The same situation can occur for all of the control types.

[0052] In one embodiment, if users have access to a custom ~~Java~~ JAVA control that they implemented or that was implemented by another developer, they can add it to a web service or another custom ~~Java~~ JAVA control. Users have access to a control if users have access to its JCS file in their project. If the control is not in their project, users can copy it to their project. If the JCS file and the associated ~~Java~~ JAVA file for the custom control users wish to use is not in their project, users can copy it to their project directory.

[0054] At their most basic, Controls users develop include a ~~Java~~ JAVA control source (JCS) file. Users can also add properties to the control by including an annotation XML file. A ~~Java~~ JAVA control source (JCS) file contains the control's logic — the code that defines what the control does. In this file users define what each of the control's methods do. Users can also define how control property values set by a developer influence the control's behavior, as in the following example.

[0055] In one embodiment, when users add a new ~~Java~~ JAVA control source file to a project, the IDE also adds a JAVA file that contains the control's public interface. Under most circumstances, users should not edit this file. By default, as users work in the JCS file, adding methods, callbacks, and implementation code, the IDE keeps the interface in sync. By way of a non-limiting example, adding an operation to the JCS will also add a corresponding method to the JAVA file. Note that the JAVA file will be kept in sync only with respect to those methods with an @common:operation annotation. This means that if users add a method to the JCS, then remove it's @common:operation annotation, the IDE will remove the method from the JAVA file.

[0056] **Figure 2** is an exemplary ~~Java~~ JAVA control source (JCS) file in an embodiment of the invention. A control implementation class Hello contains the logic for a control. The @common:control annotation tells the IDE to create and maintain this control's interface. This removes the necessity for users to do so. The control-tags annotation @jcs:control-tags associates this control source file with the annotation XML file "Hello-tags.xml" that describes the properties it exposes. The ControlContext interface provides access to aspects of a control's container, including the properties stored for the control. Control methods are operations, just as with the methods of a web service.